DTIC FILE COPY (1)

AD-A195 129

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. DCA/SW/MT-88/001a	2.	3. Recipient's Accession No.
4. Title and Subtitle Defense Communications Agency Upper Level Protocol Test System Functional Description			5. Report Date May 1988
			6.
7. Author(s)			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Defense Communications Agency Defense Communications Engineering Center Code R640 1860 Wiehle Ave. Reston, VA 22090-5500			10. Project/Task/Work Unit No.
			11. Contract(C) or Grant(G) No. (C) (G)
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered FINAL
			14.

15. Supplementary Notes

For magnetic tape, see: AD-A195 128.

16. Abstract (Limit: 200 words)

This document is part of a software package that provides the capability to conformance test the Department of Defense suite of upper level protocols including: Internet Protocol (IP) Mil-Std 1777, Transmission Control Protocol (TCP) Mil-Std 1778, File Transfer Protocol (FTP) Mil-Std 1780, Simple Mail Transfer Protocol (SMTP) Mil-Std 1781 and TELNET Protocol Mil-Std 1782.

See p 1-2

17. Document Analysis a. Descriptors

Protocol Test Systems
Conformance Testing
Department of Defense Protocol Suite

b. Identifiers/Open-Ended Terms

Internet Protocol (IP)
Transmission Control Protocol (TCP)
File Transfer Protocol (FTP)
Simple Mail Transfer Protocol (SMTP)
TELNET Protocol

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited Release	UNCLASSIFIED	33
	20. Security Class (This Page) UNCLASSIFIED	22. Price

(See ANSI-Z39.18) See Instructions on Reverse OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce

88 7 06 086

DEFENSE COMMUNICATIONS AGENCY

UPPER LEVEL PROTOCOL TEST SYSTEM

FUNCTIONAL DESCRIPTION

NTIS-$12.95

A-1 21

COPY INSPECTED 6

MAY 1988

88 7 06 086

Disclaimer Concerning Warranty and Liability

Distribution and Copyright

Comments

Comments or questions about this software product and documentation can be addressed in writing to:

DCA Code R640
1860 Wiehle Ave
Reston, VA 22090-5500
ATTN: Protocol Test System Administrator

Table of Contents

List of Figures

Figure		Page

SECTION 1 - INTRODUCTION

1.1 SCOPE AND PURPOSE OF THE DCA UPPER LEVEL PROTOCOL TEST SYSTEM

The DCA upper level protocol test system (hereafter referred to as the "test system") supports the development of Department of Defense (DOD) protocol tests to meet the need for the certification of protocols used in DoD data networks (DDN). To provide this support, the test system has been directed toward thorough functional testing of DOD standard protocols. In this role, the test system performs as a validation tool rather than as an implementation debugging facility that assists in protocol development tasks.

The test system supports all phases of certification testing, including data reduction and the development of test plans, scripts, and software specifications. The test system can test hosts or programmable devices that run protocols for hosts and terminals. The organization for whom a test is performed is responsible for implementing driver software in their machine according to existing specifications.

1.2 ORGANIZATION OF THE FUNCTIONAL DESCRIPTION

This functional description is divided into the following sections:

- o Section 1 introduces the test system design.
- o Section 2 is an overall discussion of the testing performed by the test system.
- o Section 3 identifies the major functions of the test system.

- Section 4 details the test elements currently available on the test system.

- Section 5 describes the protocol test tools available on the test system.

- A Glossary and List of References follow Section 5. The bracketed numbers in the text correlate with the List of References.

SECTION 2 - DCA UPPER LEVEL PROTOCOL TEST SYSTEM

This section on the basic philosophy of the test system describes the types of tests, the methods of testing, and the analysis of test results.

2.1 TESTING METHODS

A major goal of the test system is to test protocol implementations to determine whether they meet the functional requirements of the appropriate military standard or protocol specification. The validation tests are functional tests and not parametric studies of performance.

The packet-switched protocol architecture is designed to support a network of dissimilar hosts; subscriber systems are built on a wide variety of host computers. It is impractical for a test system facility to own every machine on which an implementation is possible, or to install those machines in the facility for testing purposes. Therefore, the test machines are left at the remote development site, and the tests are run over packet-switched facilities or dial-up lines to a test system facility.

It is not possible to completely test protocols without installing some driver software in the machine that hosts the implementation under test (IUT). Documentation exists that specify the driver software.

Protocols are specified in terms of extended or finite state machines that define the functional characteristics of interacting communicating entities. Actual protocol implementations, on the other hand, often support multiple independent communications. For example, Transmission Control Protocol (TCP) implementations should support several independent connections. The number of such connections, the way they compete for resources, and the mechanisms for keeping them

separate are implementation specific.

Testing can determine whether the independent connections interfere with each other with regard to precedence and integrity of the data flow, with each implementation having its own definition of data flow integrity. The TCP tests require that data on a connection be delivered in sequence--without alteration and without the introduction of spurious data from any other source or from the connection itself. In contrast, the Internet Protocol (IP) tests require only that the data, if they are delivered at all, be delivered to the intended upper level protocol entity.

Testing is performed under the control of a central test direction facility called the Central Driver. As Figure 2-1 shows, the site under test houses a Remote Driver that passes commands from the Central Driver to the IUT. Another kind of driver is the Slave Driver, which passes commands from the Central Driver to the Reference Implementation of the protocol (Figure 2-1). Slave Drivers, Reference Implementations, and Central Drivers are considered part of the test system. Laboratory. The Remote Driver is implemented by the vendor prior to testing and therefore is not considered part of the test system.

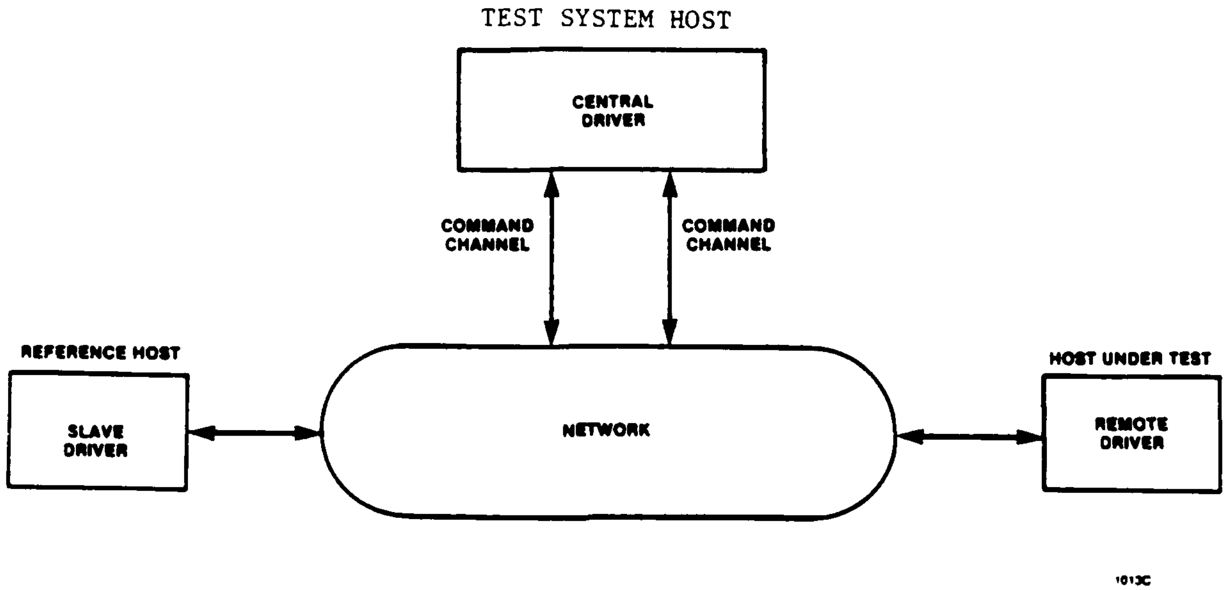

Figure 2-1. Driver-to-Driver Communications

The test system maintains comprehensive tests demonstrating that the test implementation is able to carry out normal communications with the reference version of the protocol. In addition, the test system provides facilities to generate intentionally altered protocol exchanges. This process validates whether the IUT is robust enough to receive incorrect sequences, reject the types of improper sequences it was designed to reject, and to generate the appropriate responses to these improper sequences (i.e., error codes) to its peer.
In order to ensure the complete testing of an IUT's IP without the possible intervention of intermediate IP (e.g. gateways), all IP testing is performed on the same proximate network connected to the test system facility.

2.1.1 Top-Down Testing

Top-down testing consists of testing the protocol from the vantage point of an Upper Layer Protocol (ULP). The higher level entities expect a protocol implementation to provide a set of services. To use the services, these entities issue requests in the form of interface events, with parameters requesting that the protocol provide some specified service.

Figure 2-2 shows a generic model of the ULP interface. The ULP makes a request for a particular service offered by the protocol under test. The protocol under test then returns a response to that request. This generic model assumes that all requests made by the ULP are legitimate; therefore, the response codes indicate only the success or failure of the protocol under test in its attempt to provide the service. To fully test a protocol's upper layer interface, the model must be expanded (Figure 2-3). The test model indicates the possibility that legitimate

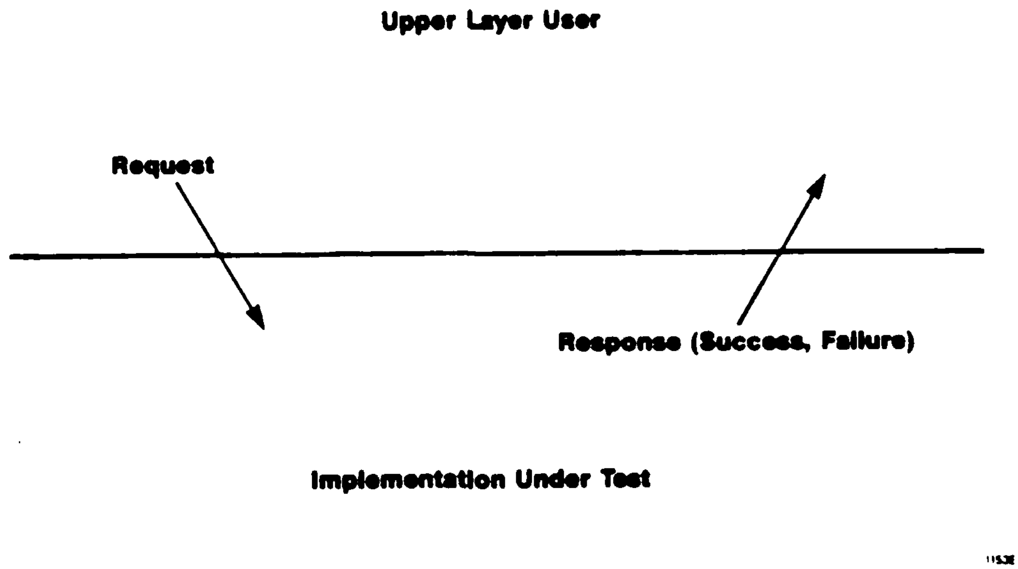

Figure 2-2. ULP Interface

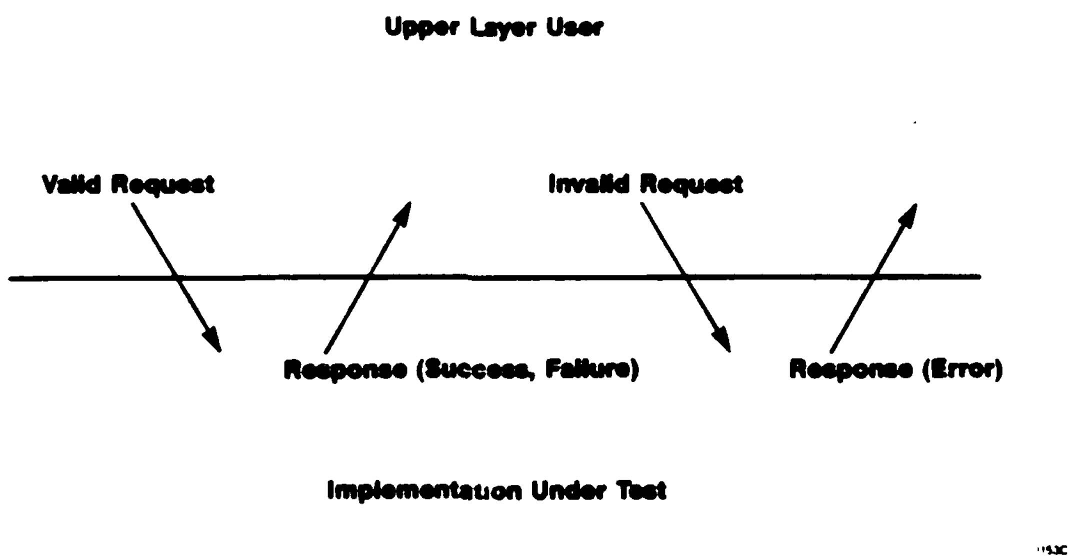

Figure 2-3. ULP Test Interface

and illegitimate requests might be made by the ULP. As Figure 2-3 shows, the protocol under test must also be able to handle these invalid requests properly by providing the ULP with an error response.

These two requests-responses require two different types of tests. Functional testing is used for the valid request-response cases. The invalid request-response cases are tested by error-rejection tests.

2.1.1.1 Functional Testing

A protocol implementation must provide all required services in the protocol's specification. The implementation also must support all the user-selectable service options that it claims to provide. Testing consists of the following:

1. The services provided by the protocol are identified for each protocol implementation under test (IUT). Most protocol specifications provide a core of service requirements and a large list of optional services that may be implemented. Tests have been developed for each service requirement identified for a specific protocol, including options.

2. These tests have been arranged into batteries of tests from which the test operator is able to select the tests to be run. For example, only seven of the commands specified in the File Transfer Protocol (FTP) are identified as mandatory. The battery of mandatory tests are the first ones presented to the test operator for selection. The optional tests then are divided into three categories: data transfer, file management, and three-party manipulation. For the optional services, the test operator can select the battery of tests for each of the categories. Once the test operator selects the

particular battery, the Central Driver initiates the test by opening connections to the various drivers and sending them the commands. For optional services, "not-implemented" is a perfectly valid response to a particular command from the IUT's Remote Driver. The Central Driver will note this response in a disk file generally known as a "log" file, and then will skip to the next test.

2.1.1.2 Error-Rejection Testing

In addition to responding to requests for services that are not implemented, the protocol must be able to reject invalid parameters and ULP requests that occur at the wrong time. The error-rejection tests were built by determining what events and parameters are invalid for any given state of the protocol and then creating the test scenarios that would produce those error conditions.

Testing precludes instrumentation in the IUT. Therefore, the IUT's state must be deduced from observing any activity at the upper level interface and by sending additional message sequences that have predictably different results--depending on whether or not the implementation under test has changed states.

2.1.2 Bottom-Up Testing

Bottom-up testing consists of evaluating how the IUT responds to control information from a peer entity.

Bottom-up testing strategy is much the same as that for top-down testing. The main differences have to do with what software is used to do the testing and where that software resides in the testing architecture. Top-down testing involves running software that accesses the IUT at its upper layer interface. In bottom-up testing the peer messages must be modified and/or duplicated in a

manner appropriate to the current state of the protocol entity generating the message. This capability has been achieved by modifying the standard version of the protocol to allow incorrect Protocol Data Units (PDUs), improper sequences of PDU's, and other erroneous PDU's to be produced upon demand. In this document, the modified standard version is referred to as the Reference Implementation. Figure 2-4 illustrates the typical test configuration for bottom-up testing.

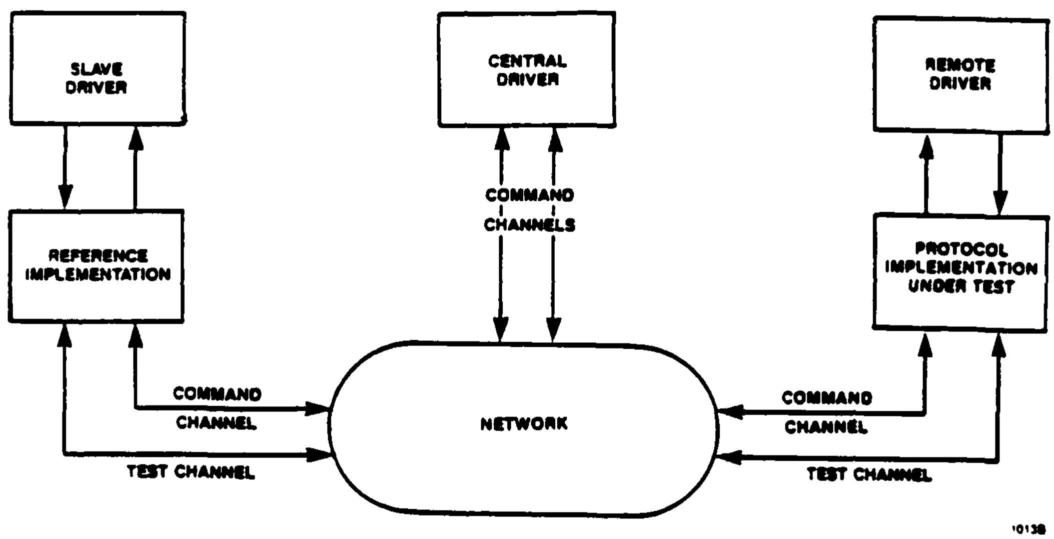

Figure 2-4. Bottom-Up Testing

2.2 TEST MATERIALS

To shorten tests and improve their repeatability, the protocol tests have been automated as much as possible. A series of scripts has been prepared to direct the test drivers. When used together to test a particular protocol under a particular set of conditions, this collection of scripts is called a scenario. The scenario is compiled to form a Central Driver that is capable of conducting the created test situation. A typical scenario consists of:

- Set-up commands;
- Synchronization commands that begin the test and coordinate the passing of commands to the test drivers;
- Commands that analyze the results returned from the test drivers; and
- Test completion commands.

The test scripts and scenarios have been written in the Test Scenario Language (TSL) created for the test system. The language is fully defined in the TSL Programmer's Manual [1]; and briefly described in this document in Section 4.2. TSL provides mechanisms for the scenario writer to invoke protocol service primitives by sending protocol-specific commands (PCMDS) to the appropriate driver. PCMDS are created from the set of protocol commands defined in each specification. A Remote Driver specification has been created for each protocol being tested. The specification defines the PCMDS for that particular protocol and details to the Remote Driver the expected actions to be taken when a command is issued.

TSL contains synchronization commands (WAIT) that will suspend the test procedure until the drivers respond. TSL then provides the analysis commands (IF..., CONTAINS..., etc.) that direct the scenario to perform the next test or to skip to another test, depending on the response to the last command. Finally, TSL

contains the vital logging functions necessary for presenting test results, as well as for writing the raw data in the responses into a disk file.

SECTION 3 - DCA UPPER LEVEL PROTOCOL TEST SYSTEM
TEST SYSTEM FUNCTIONS

This section describes the major test system functions.

3.1 CONDUCT TESTS

The test system performs the testing necessary for generating validation test reports, including tests to ensure that:

- The protocol implementation performs all the functions identified as services to upper level protocols.
- The quality of services provided to the Upper Layer Protocols corresponds to those defined in the protocol standard.
- The protocol responds correctly to flawed or erroneous data, where the notion of correct response is protocol dependent.
- The protocol does not have side effects that violate the specification or standard, or the intent of the specification or standard.

During the testing, the test system records detailed test data and evaluations. These data are used to generate detailed reports that allow independent validation of test accuracy and reliability. The test reports can be made available in either hard copy or on magnetic media. These results are maintained in on-line files that are controlled by the configuration management system.

SECTION 4 - DCA UPPER LEVEL PROTOCOL TEST SYSTEM FUNCTIONAL ELEMENTS

This section details the functional elements currently available in the test system. The next section illustrates the use of those elements in the various operational protocol test facilities.

4.1 TEST DRIVERS

In addition to providing Central Drivers as the single control facility for coordinating and conducting tests of implemented protocols, the test system provides all the Slave Drivers needed for testing a particular protocol. The test system also provides the specification for the Remote Drivers, but it is up to each vender to provide his own Remote Driver.

Although its generic role is to interface between the Central Driver and the particular protocol implementation, the role of the driver in the testing has taken on many functions. For testing Application Protocols, a vital element is to have a Remote Driver supported by the host under test. Because he can assume this condition exists, the person creating the tests is free to define a constant interface without having to worry about the peculiarities of a particular host command language. This capability is very useful when one considers the number of different hosts that will be tested and the number of different user command languages each host supports. Also, in testing Application Protocols, various tests require information that is unique to a file system or operating system. The Remote Driver specification will provide the general commands to request this specialized information. Each Remote Driver must then be able to interpret these commands and send to the Central Driver the appropriate data. For example, the File Transfer Protocol (FTP) Remote Driver must be able to run a validation algorithm across a designated file and return the resulting value to the Central Driver. How the Remote Driver accesses the particular file and

runs the algorithm is strictly an implementation issue.

4.2 TEST SCENARIO LANGUAGE (TSL)

The test system uses TSL, a high-order test development language that provides commands for test initialization, test control and coordination, test results analysis and logging, and test completion. The initialization statements allow the user to open command connections from the Central Driver to either the Remote or the Slave Driver. When the command connections are established, TSL provides commands for sending protocol- or driver-specific commands and for receiving the results of the command. TSL statements allow the user to analyze the response for correctness and validity. TSL-supplied conditional statements can be used with evaluation statements to direct the flow of the test through the various scripts available at execution. Perhaps the most important test language statements are the logging statements, because they let the user know the status of the ongoing test. TSL provides a variety of logging statements that allow the scriptwriter to direct the output information to file storage media or to both an interactive display and storage media.

Associated with the TSL is a compiler and run-time support package. Figure 4-1 illustrates the TSL compiler processing commands into C statements and calling the C compiler for object code generation. The object code generated is then linked with a run-time support package that provides the procedures for implementing the TSL commands. The compilation and linkage processes have been incorporated into a C-shell script file so that the user sees only the equivalent of a TSL compiler.

4.3 REFERENCE IMPLEMENTATIONS

The test system currently maintains "altered" implementations (References) of Military Standard protocols. These references generate valid and invalid protocol packets under direction of

the associated Slave Driver. The references support all the functional requirements and options of the particular protocol and can be used for the top-down testing described in Section 2. In addition, these references can corrupt packets and therefore are used for bottom-up testing, also discussed in Section 2. Reference Implementations exist for IP, TELNET, FTP, TCP, and Simple Mail Transfer Protocol (SMTP).

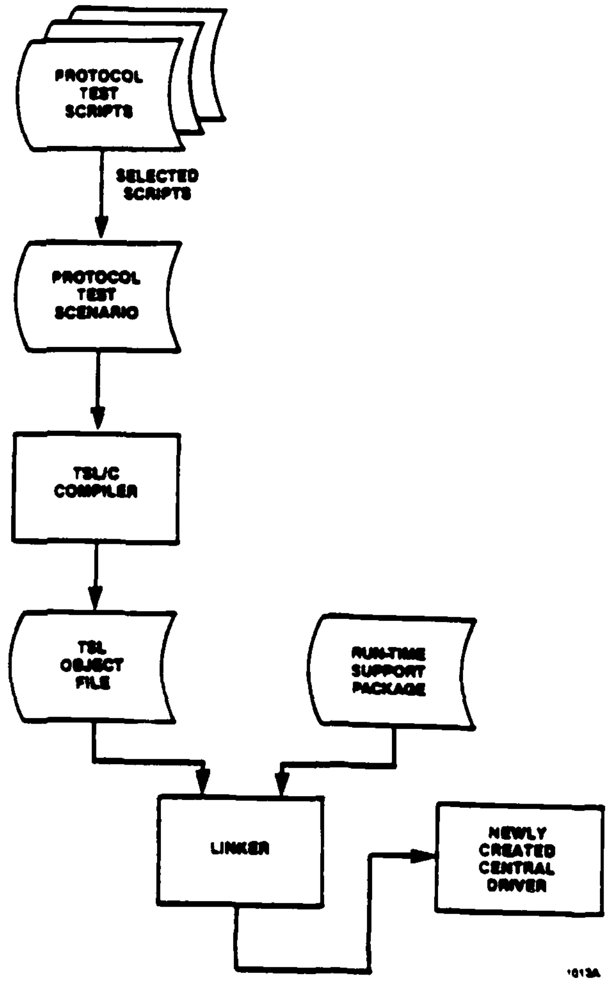

Figure 4-1. TSL Compilation Process

4.4 SOFTWARE SUPPORT TOOLS

The test system provides a consistant interface to the operator through the use of the Man-Machine Interface (MMI) and the Report Generator (RG). The Man-Machine Interface provides the operator with a single execution methodology for all tests. While the Report Generator provides a consistant view of the results of all tests.

4.4.1 Man-Machine Interface

The MMI provides the test operators with an easily understood menu-oriented interface that allows test execution and results analysis to be performed in a flexible and consistant manner for all protocols supported by the test system. The MMI provides a high-level of abstraction of the testing process so that much of the underlying complexity of the process is hidden from the operators. Yet the MMI is "user-friendly" enough that the novice operators can very quickly become proficient users of the system. In addition to being user-friendly, the MMI is flexible enough to provide the same consistant interface to any new tools that will be added to the test system.

4.4.2 Report Generator

The RG allows the inspection, summary, and analysis of test results to be performed in a simple and consistant manner for all protocols supported by the test system. The RG's operator interface is easy to use and provides the reporting mechanisms that have been found to have the most utility on the test system. It hides the underlying file structure and file access mechanisms from the operator. Like the MMI the Report Generator was designed to be flexible enough to allow the easy addition of new tests and reports.

4.5 CONFIGURATION MANAGEMENT SUPPORT TOOLS

The test system uses the Source Code Control System (SCCS) for maintaining the integrity of the baseline support software. By providing library mechanisms for checking software in and out, SCCS prevents programmers from destroying each other's work. In addition, SCCS provides a version system so that historical versions of released software can be regenerated, if necessary. All the programming modules and scripts are placed into the SCCS.

SECTION 5 - DCA UPPER LEVEL PROTOCOL TEST SYSTEM

FUNCTIONAL PROTOCOL TESTS

5.1 INTERNET PROTOCOL (IP) TEST ARCHITECTURE

Figure 5-1 illustrates the IP testing mechanisms. In this architecture the Central Driver communicates only with the Slave Driver since the availability of TCP on the host under test can not be assumed. Likewise the Slave Driver communicates directly with the IP Reference advoiding any interaction with the TCP interface.

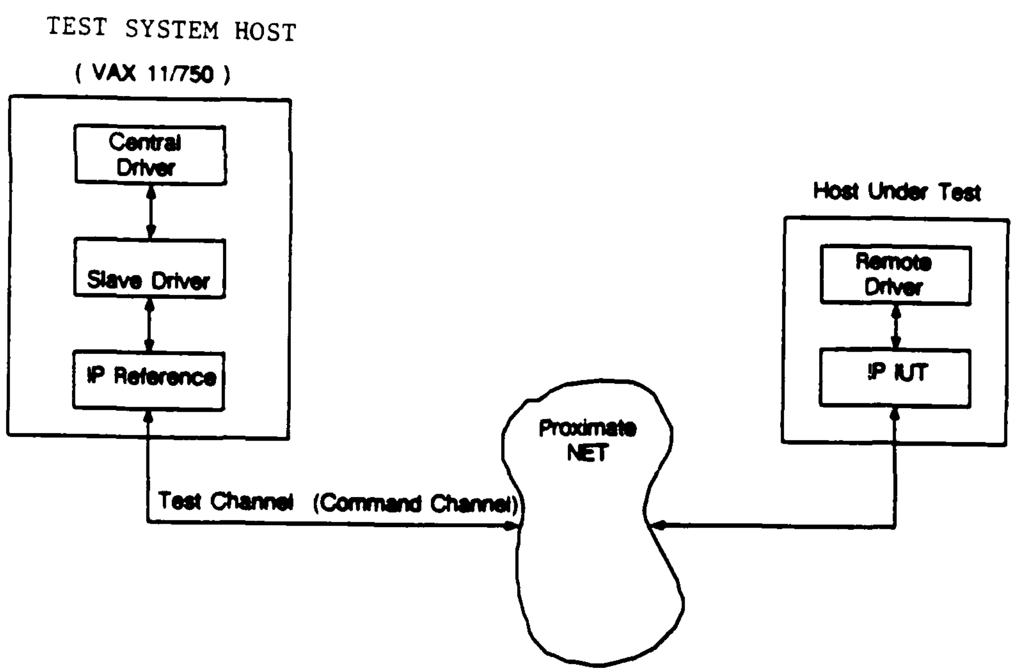

Figure 5-1. Internet Protocol (IP) Test Architecture

As the figure shows, the IUT must be on the same proximate network as the test system. This connectivity ensures that there are not any network-caused packet deletions, a circumstance that is permissible on the operational INTERNET System. In this configuration, the Reference can send invalid IP packets to the IUT and be sure it is the IUT's IP that either accepts or rejects the packets, and not some intervening gateway. All ICMP response messages received by the Reference are evaluated as if the IUT sent them.

Since the normal TCP command channel is not available for IP testing, the commands are sent as part of the Reference's IP packets data section. Therefore, to perform top-down testing, the Remote Driver shown above the IP IUT must be able to accept the incoming IP packet and interpret the commands found in the data section of that packet according to the IP Remote Driver Specification [2]. These commands will direct the Remote Driver to create IP packets of various lengths with a variety of headers. Similarly, the top-down tests direct the Remote Driver to vary the header information so that the IUT's ability to handle precedence, and various routing directions will be demonstrated.

The bottom-up tests verify the IUT's ability to withstand incorrect or inappropriate IP packets from the Reference Implementation. Valid and invalid IP fragments are sent to test the IUT's ability to reassemble the packet correctly, or to discard the packet pieces after timing out. All test results are logged to a file which can be interpreted by the Report Generator to determine the IP IUT's success or failure to properly execute responses. Additional test capability information on IP testing is found in the IP Test Manual [3].

5.2 TRANSMISSION CONTROL PROTOCOL (TCP) TESTING

5.2.1 Functional Overview

Figure 5-2 illustrates the system functions that comprise the TCP test suite in the test system.

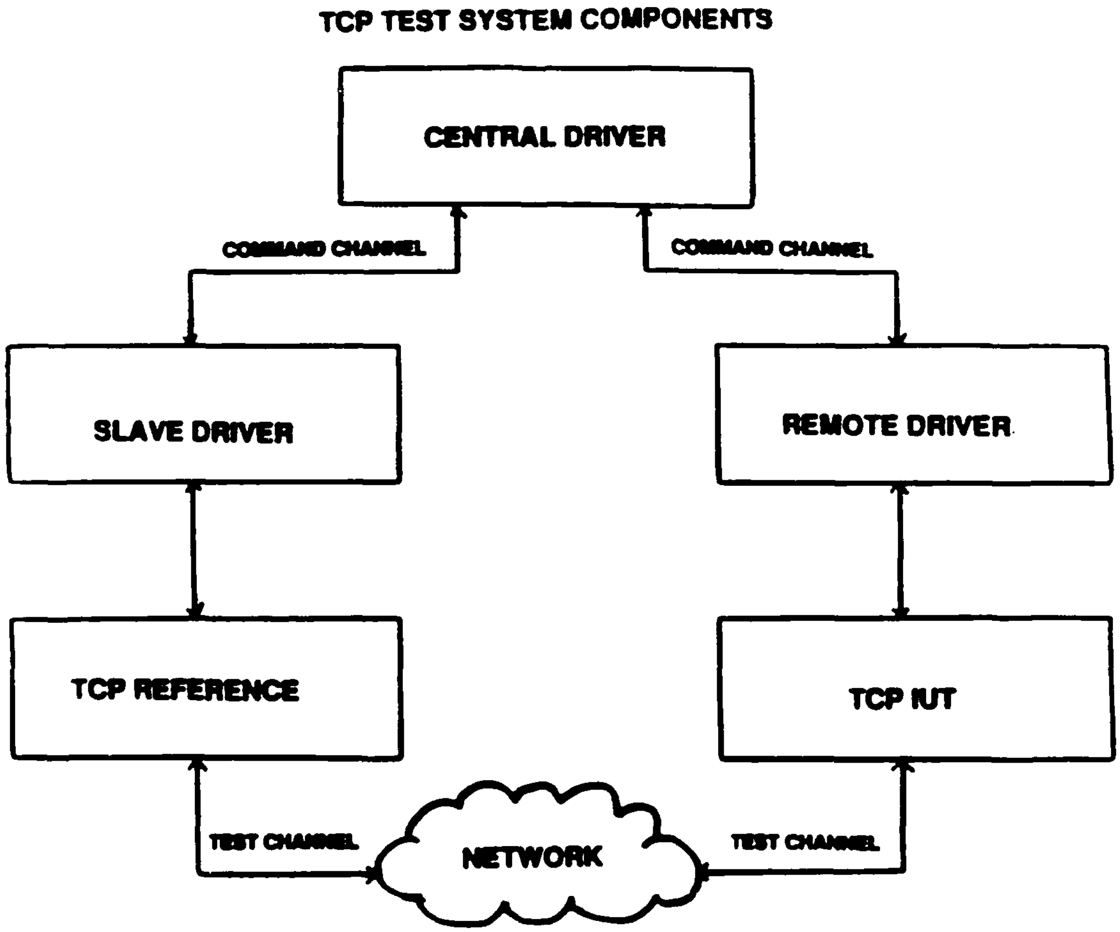

Figure 5-2. Transmission Control Protocol (TCP) Test Architecture

This Central Driver acts as the controller in conducting the tests. It either establishes or causes to have established all the communications needed for testing. It interacts with the Slave Driver and Remote Driver in performing the tests. This interaction is a process of analyzing the data returned from the drivers and deciding which section of the test scenario to execute next. In addition the Central Driver performs the logging functions for all results obtained during the testing process.

This Slave Driver is the intermediary process between the Central Driver and the Reference Implementation. It performs two major actions: Translation of Central Driver commands into the proper TCP reference requests and translation of the raw data returned from the TCP reference into Central Driver messages.

This Remote Driver is the intermediary process between the Central Driver and the Implementation Under Test. Its primary function is to translate user-level TCP service requests into the appropriate TCP requests for the IUT. It also relays back to the Central Driver all of the responses received from the IUT. This function is required for performing all Top-Down tests and for recording any error conditions created by the Bottom-Up testing.

This Reference Implementation is a highly instrumented version of TCP that performs all of the services defined in MIL-STD 1778. Under Central Driver control the reference also can send invalid TCP segments to the IUT as part of the Bottom-Up testing. Because of the real-time nature of TCP, it cannot be presumed that the Central Driver can interact quickly enough in a test situation not to bias the test results. Therefore, the reference only receives test parameter settings prior to test activation and then carries out the testing according to those settings. The parameters that can be set govern the following functions in TCP testing:

(1) The ability to reset all of the TCP parameters to their initial condition;

(2) The ability for TCP to deliberately send data out of sequence;

(3) The ability for TCP to deliberately send segments, each of which contain data that had already been transmitted;

(4) The ability for TCP to deliberately send segments out of order;

(5) The ability for TCP to set the maximum segment size option to a specified value;

(6) The ability for TCP to falsify the sending port number on various segments in the same data stream.

5.2.2 Operational Overview

Figure 5-3 illustrates the operational implementation of the TCP test suite. The Central driver, the Slave Driver and the Reference Implementation reside on the host. The Remote Driver and the IUT reside on the Host under Test which is assumed to be reachable via a network interface. The command channel goes from the Central Driver to the Remote Driver while the test channels are created between the Reference Implementation and the IUT. Because the command channel does run over a TCP connection, the test system assumes that the IUT is minimally capable of sending and receiving data.

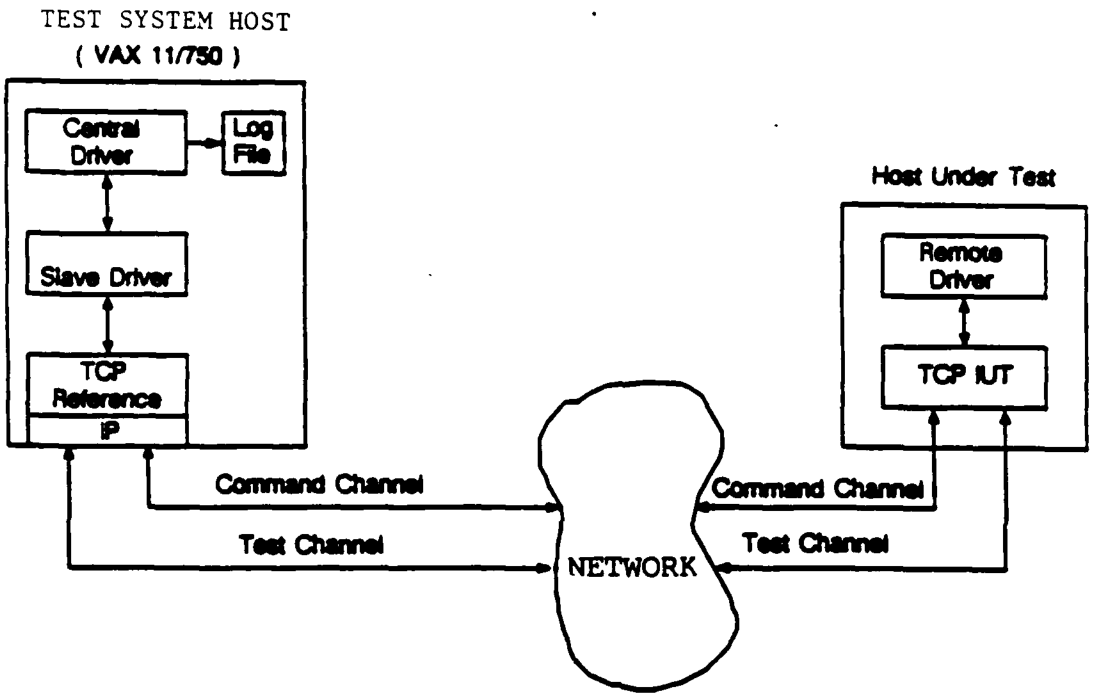

Figure 5-3. Operational TCP Test Architecture

5.3 TCP/IP Testing (Tightly-coupled implementations)

The test system provides the ability to test versions of TCP/IP that can not separate the interface (expose) between the two protocols. This TCP/IP implementation is known as tightly-coupled TCP/IP. The functional components for TCP/IP testing are similar to those for TCP testing, with the reference being a combination of TCP and IP in order to perform the testing. The operational testing that is restricted to TCP and IP tests does not require the exposure of the common interface. For example, the Top Down IP tests can not be done since they assume access to the IP Upper Interface. The Bottom-Up TCP tests can be executed only after IP has been tested to some degree. Figure 5-4 illustrates the operational TCP/IP test architecture. The reference implementation is a TCP/IP implementation, as indicated, while the Slave Driver is the same as the TCP Slave Driver with the ability to parse data coming directly from IP.

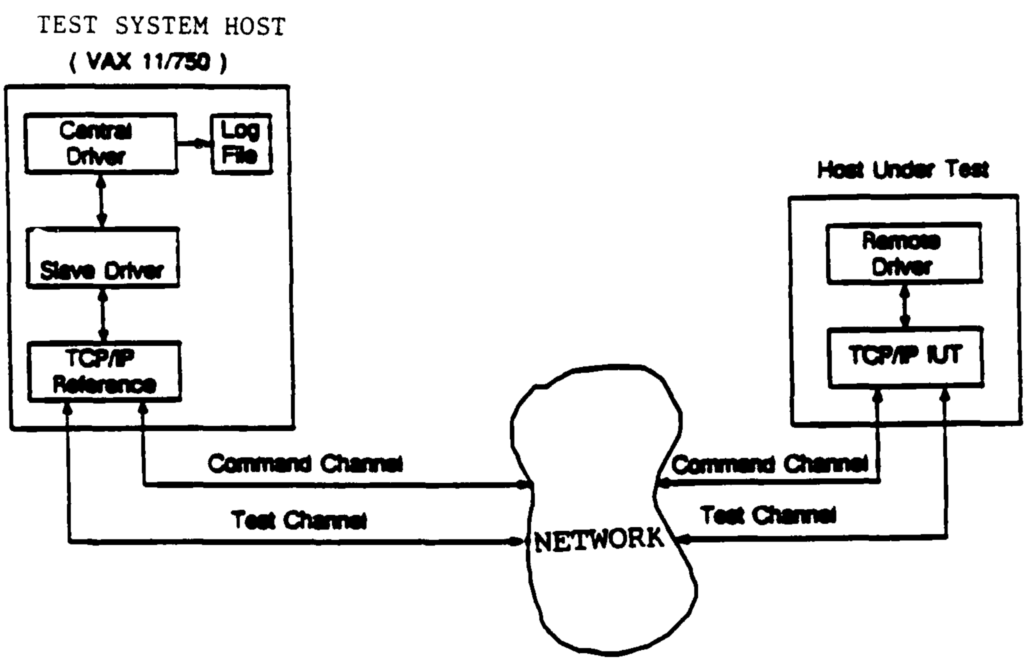

Figure 5-4 Operational TCP/IP Test Architecture

5.4 APPLICATION PROTOCOL TESTING

5.4.1 FTP and Telnet

Figure 5-5 shows the typical Application Protocol test architecture. The Central Driver has been created from a scenario, compiled and linked with the run-time package. The Remote Driver and test system processes have been initiated, and they have created interprocess links with their respective protocol implementations. The figure illustrates the Slave Driver and the Reference Implementation housed in the host.

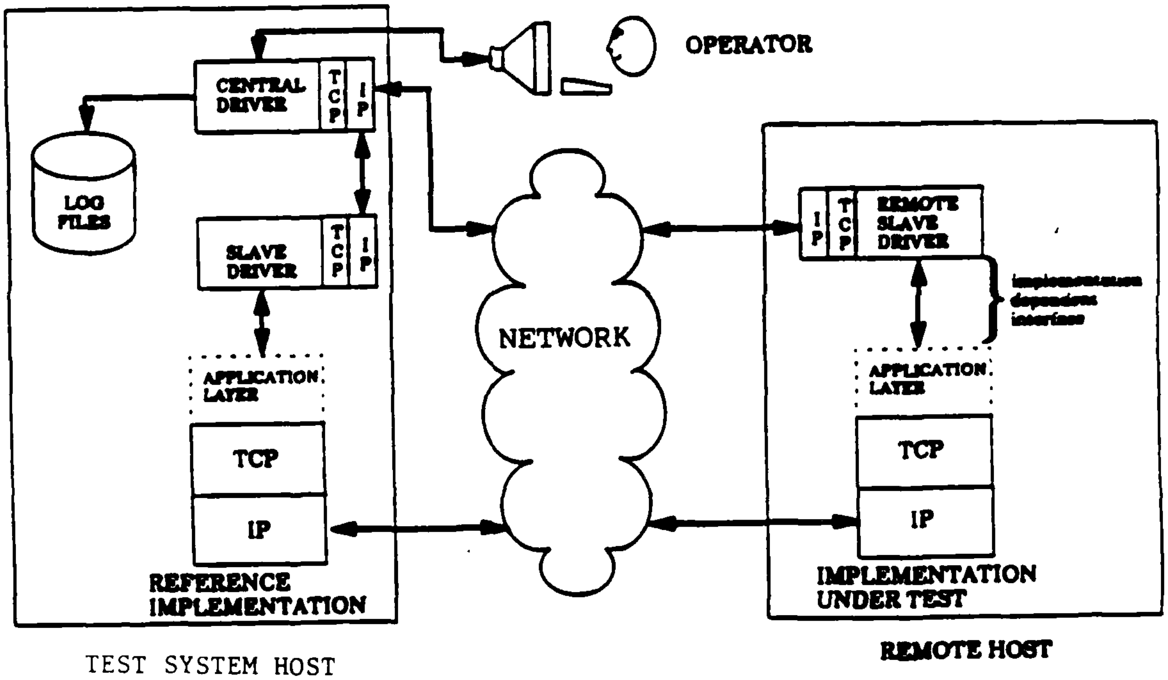

Figure 5-5. Application Protocol Test Architecture

Before an implementation can become a Reference, it must be able to meet the functional requirements of the protocol standard. In this case, the FTP and TELNET User and Server Implementations were enhanced to provide all of the required and optional services. The standard implementations were customized to provide the additional testing information specific to the protocol. For example, the TELNET Reference Implementation translates internal protocol commands into ASCII strings for output to the Slave Driver. Also, the TELNET Server Reference Implementation was enhanced to initiate option negotiation sequences in addition to the normal response mechanisms. This additional option negotiation ability allows the test operator to do bottom-up testing by changing the test environment as it relates to full- or half-duplex data transfer and other options from the Reference Server.

The drivers in Application Protocol testing may be called upon to perform additional tasks beyond interacting with the protocol implementation. For example, the FTP Drivers must be able to perform validation algorithms on data files to ensure the files were created correctly.

5.4.2 Simple Mail Transfer Protocol (SMTP)

The SMTP test architecture is consistent with TSL architecture illustrated in figure 5-5. In this test environment the Central Driver has a different relationship with the Slave Driver and the Reference Implementation, in that the Central Driver provides the Reference with the exact response code for each transition in the protocol. This permits the Script Writer to vary the References appearance to the IUT from test to test, depending upon the response received from the IUT.

The Remote Driver for SMTP does not communicate directly with the SMTP IUT. Instead it places "mail" messages in the mail queue which are to be transmitted by the IUT, and it reads the mail

messages sent from the Reference Implementation and transmits the content back to the Central Driver.

APPENDIX A - GLOSSARY

Central Driver - the controlling test driver and the interface to the operator's console. It coordinates all the test drivers.

Environment - includes an operating system and other software that provides computational support services to protocol entities.

File Transfer Protocol (FTP) - a DOD standard protocol used to provide reliable transfer of files between host machines, usually geographically separated.

Gateway - a device or pairs of devices that interconnect two or more subnetworks to enable the passage of data from one subnetwork to another.

Internet - the term used to identify a collection of wide area networks and local area networks, each concatenated to the others through a system of gateways.

Internet Control Message Protocol (ICMP) - used by IP to send control messages to the other IP entities. ICMP messages are used to control routing and congestion and to support other internet housekeeping chores.

Internet Protocol (IP) - one of the DOD standard protocols. It provides a connectionless datagram service from sources to destinations throughout the internet, and a virtual network address space. Sources and destinations are hosts located on either the same subnetwork or connected subnetworks. The virtual network address space allows any host to address any other host anywhere in the internet.

Slave Driver - translates Central Driver commands into the appropriate interface action requests and passes those requests on to the Reference Implementation.

Protocol - a set of rules governing the exchange of information. Protocols contain conventions managing data flow for data encoding and error handling.

Protocol Data Unit (PDU) - a message that is exchanged between peer protocol entities. A protocol data unit may contain both control and data information.

Remote Test Driver - translates Central Driver commands into the appropriate interface action requests and passes those requests on to the implementation under test.

Simple Mail Transfer Protocol (SMTP) - a standard DOD protocol designed to send and receive mail.

Telnet - the DOD standard protocol for providing Network Virtual Terminal service.

Transmission Control Protocol (TCP) - the DOD standard connection-oriented transport protocol. TCP provides a reliable, sequenced end-to-end stream service.

APPENDIX B - References

1. Test Scenario Language Programmers Manual, (developed under DCA contract DCA 100-83-C-0064), 15 July 1988.

2. DCA Upper Level Protocol Test System, Internet Protocol MIL-STD-1777 Remote Driver Specification, May 1988.

3. Laboratory Implementation Plan, 31 January 1985 (developed under DCA contract DCA 100-83-C-0064).

4. Laboratory Specification, 10 August 1984 (developed under DCA contract DCA 100-83-C-0064).

5. Host-to-Front-End Protocol Specification, 19 July 1985 (developed under DCA contract DCA 100-83-C-0064).

6. Transmission Control Protocol, MIL-STD-1778, 12 August 1983.

7. Internet Protocol, MIL-STD-1777, 12 August 1983.

8. File Transfer Protocol, MIL-STD-1780, 10 May 1984.

9. Simple Mail Transfer Protocol, MIL-STD-1781, 10 May 1984.

10 Telnet Protocol, MIL-STD-1782, 10 May 1984.